Tokenizer Generator

NLP Course Task 1

Under the Supervision of Dr. Shamsfard

Rashin Rahnamoun

October 18, 2024

Abstract

The process of tokenization plays a critical role in natural language processing (NLP), where it serves as the foundation for text analysis, feature extraction, and model development. This report presents the design and implementation of a **Tokenizer Generator**,a language-independent system that generates the most effective tokenizer algorithm based on the given input, regardless of its nature or context. a system capable of generating a diverse array of tokenizers tailored for various linguistic tasks. I investigate multiple tokenization strategies, including word, sentence, and character-based approaches, as well as more sophisticated methods like regex-based, context-sensitive, and frequency-based tokenizers. The study also includes a short literature review of existing Persian tokenizers and their methodologies, examining complex Persian texts with different challenges to generate the best possible tokenizer for each text and task. Our evaluation covers a broad spectrum of tokenization techniques, as reflected in precision, recall, F1-score, and accuracy, highlighting their effectiveness across different tasks and datasets. A detailed performance analysis is provided to facilitate the selection of appropriate tokenizers for specific NLP applications. Nearly 200 generated tokenizers' algorithms were tested.

1 Recent Works

Tokenization plays a significant role in the process of lexical analysis. Tokens become the input for other natural language processing tasks, like semantic parsing and language modeling. Natural Language Processing in Persian is challenging due to Persian's exceptional cases, such as half-spaces. Thus, it is crucial to have a precise tokenizer for Persian.

This article [1] provides a novel work by introducing the most widely used tokenizers for Persian and comparing and evaluating their performance on Persian texts using a simple algorithm with a pre-tagged Persian dependency dataset.

After evaluating tokenizers with the F1-Score, the hybrid version of the Farsi Verb and Hazm with bounded morphemes fixing showed the best performance with an F1 score of 98.97%.

A paper [2] introduced DadmaTools, an open-source Python Natural Language Processing toolkit which also introduces a tokenizer for persian language. Tokenization is an understudied and often neglected component of modern large language models (LLMs). Most published works use a single tokenizer for all experiments, often borrowed from another model, without performing ablations or analysis to optimize tokenization. Moreover, the tokenizer is generally kept unchanged when fine-tuning a base model.

In this work [3] has showed that the size, pre-tokenization regular expression, and training data of a tokenizer can significantly impact the model's generation speed, effective context size, memory usage, and downstream performance. it trained specialized Byte-Pair Encoding code tokenizers and conduct extensive ablations on the impact of tokenizer design on the performance of LLMs for code generation tasks such as HumanEval and MBPP. Furthermore, this work provides recommendations for tokenizer hyper-parameter selection and switching the tokenizer in a pre-trained LLM. Our experiments include models trained from scratch and from pre-trained models, verifying their applicability to a wide range of use cases.

the paper found that when fine-tuning on more than 50 billion tokens, it can specialize the tokenizer of a pre-trained LLM to obtain large gains in generation speed and effective context size for the Persian language. Here's a paraphrased version of your text suitable for the "Recent Works" section of your paper:

Recent advancements in language models have revealed impressive performance in multilingual contexts, even in cases where they were not specifically trained for such tasks. Nonetheless, concerns remain about the output quality across different languages. This research [4] underscores how variations in the treatment of languages occur at the tokenization stage, long before the model is utilized. Notably, the same text translated into various languages can result in significantly different tokenization lengths, with disparities of up to 15 times observed in certain situations.

Such inconsistencies are evident even with tokenizers designed explicitly for multilingual applications. Additionally, character-level and byte-level tokenization methods exhibit differences in encoding lengths exceeding four times for specific language pairs. These variations contribute to unequal treatment of various language communities, impacting the costs associated with accessing commercial language services, as well as processing time, latency, and the volume of content that can be provided as context to the models. Therefore, this work advocates for the development of future language models that utilize multilingually fair subword tokenizers.

2 Methodology

Let $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ be a set of n algorithms. For each algorithm $A_i \in \mathcal{A}$, we define a performance function:

$$P(A_i) : A_i \to \mathbb{R}$$

which maps each algorithm to a real-valued performance score. Our objective is to maximize the performance score over the set of algorithms, selecting the top N algorithms:

$$\mathcal{A}_{\text{top}} = \underset{A_i \in \mathcal{A}}{\text{argmax}}\, P(A_i) \quad \text{for } i = 1, \ldots, N.$$

Given the selected top N algorithms, we then apply a refinement procedure to each algorithm:

$$A_i' = \text{Refine}(A_i), \quad A_i' = A_i + \Delta A_i,$$

where ΔA_i represents an incremental improvement based on parameter tuning. The refinement process can be modeled as an optimization problem:

$$\Delta A_i = \underset{\Delta A_i}{\text{argmin}}\, L(A_i'; \theta),$$

where L is a loss function, and θ represents the set of tunable parameters.

Next, the refined algorithms are combined into a final algorithm A_{combined} using a weighted sum:

$$A_{\text{combined}} = \sum_{i=1}^{N} \alpha_i A_i', \quad \alpha_i \in \mathbb{R}, \sum_{i=1}^{N} \alpha_i = 1,$$

where α_i are weighting coefficients representing the contribution of each algorithm to the final solution.

2.1 Performance Evaluation

The performance of the combined algorithm A_{combined} is then evaluated using the overall performance function:

$$P(A_{\text{combined}}) = f\left(\sum_{i=1}^{N} \alpha_i P(A_i')\right),$$

where f is a function that aggregates the individual performance scores. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ be a set of n tokenizer algorithms generated with GPT4-O. These algorithms are evaluated using two test cases: (1) manually created test cases and (2) automatically generated test cases using a pipeline with GPT4-O.

For each algorithm A_i, the evaluation is based on metrics including accuracy, precision, recall, and F1 score. Let the manually generated tokenized dictionary be $\mathcal{D}_{\text{manual}}$ and the GPT4-O generated dictionary be $\mathcal{D}_{\text{GPT4-O}}$.

2.2 Evaluation Metrics

The following metrics are used to evaluate the performance of each algorithm:

Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where:

- TP = True Positives (correctly identified tokens),
- TN = True Negatives (correctly rejected tokens),
- FP = False Positives (incorrectly accepted tokens),
- FN = False Negatives (missed tokens).

Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1 Score

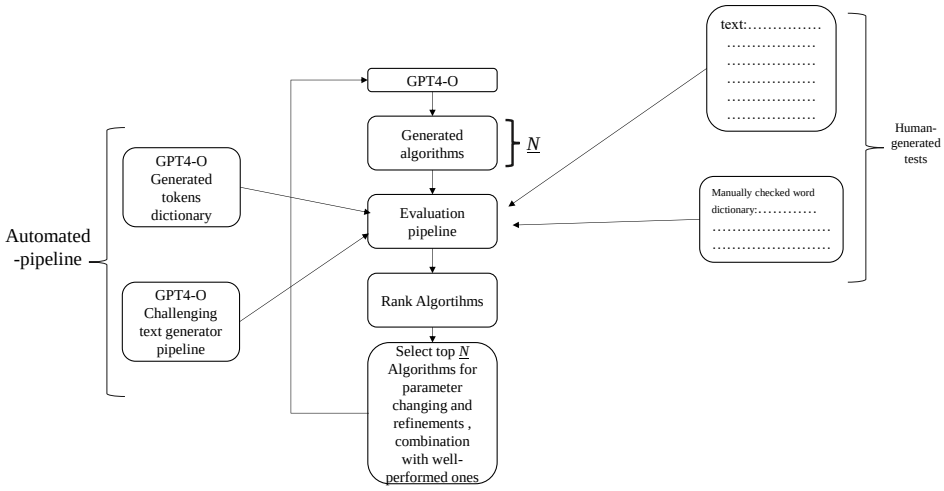
$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The evaluation pipeline ranks the top algorithms based on the F1 score and accuracy. The tokenization output for each algorithm is compared with the manually generated dictionary $\mathcal{D}_{\text{manual}}$ or the GPT4-O generated dictionary $\mathcal{D}_{\text{GPT4-O}}$.

Algorithm 1 Top N Algorithm Selection, Refinement, and Combination

$\mathcal{A} \leftarrow \{A_1, A_2, \ldots, A_n\}$ ▷ Initialize the set of algorithms
for each $A_i \in \mathcal{A}$ **do**
    $P(A_i) \leftarrow$ Evaluate performance of A_i
end for
$\mathcal{A}_{\text{top}} \leftarrow \underset{A_i \in \mathcal{A}}{\text{argmax}}\, P(A_i)$ ▷ Select top N algorithms
for each $A_i \in \mathcal{A}_{\text{top}}$ **do**
    $A_i' \leftarrow A_i + \Delta A_i$ ▷ Refine each algorithm by tuning parameters
end for
$A_{\text{combined}} \leftarrow \sum_{i=1}^{N} \alpha_i A_i'$ ▷ Combine refined algorithms with weights α_i
return A_{combined}

3 Pipeline Illustration



4 Results

For the initial testing, randomly selected Persian Wikipedia pages were manually tokenized. The results are as follows, with well-known Persian tokenizers also included in the evaluation. 146 tokenizers unique algorithms were generated and tested

Table 1: Tokenizer ID and Name Mapping

ID	Tokenizer Name
1	basic_word_tokenizer
2	regex_persian_word_tokenizer
3	whitespace_tokenizer
4	stopword_removal_tokenizer
5	stemmer_tokenizer
6	punctuation_removal_tokenizer
7	length_based_tokenizer
8	unique_word_tokenizer
9	number_aware_tokenizer
10	long_words_tokenizer
11	short_word_tokenizer
12	reverse_word_tokenizer

ID	Tokenizer Name
13	specific_word_removal_tokenizer
14	frequency_tokenizer
15	sentence_based_word_tokenizer
16	bigram_tokenizer
17	trigram_tokenizer
18	ngram_tokenizer
19	persian_and_punctuation_tokenizer
20	sentence_then_word_tokenizer
21	specific_length_tokenizer
23	ignore_digits_tokenizer
24	frequency_count_tokenizer
25	ignore_short_words_tokenizer
26	reverse_each_word_tokenizer
27	startswith_tokenizer
28	endswith_tokenizer
30	ignore_punctuation_tokenizer
31	long_sentence_tokenizer
35	persian_and_digits_tokenizer
36	remove_short_words_tokenizer
37	frequent_words_tokenizer
40	first_char_length_tokenizer
41	alphabetical_tokenizer
42	repeated_words_tokenizer
43	case_insensitive_tokenizer
44	ignore_special_char_tokenizer
45	endswith_specific_char_tokenizer
46	short_sentence_tokenizer
48	character_tokenizer
49	count_vectorizer_tokenizer
50	normalized_tokenizer
51	ParsBert_tokenizer
52	exclamatory_sentence_tokenizer
53	contextual_word_tokenizer
54	adverb_tokenizer
55	interjection_tokenizer
57	word_count_tokenizer
59	phrase_tokenizer
61	content_word_tokenizer
62	repeated_character_tokenizer
63	frequency_filter_tokenizer
66	random_sampling_tokenizer
67	common_prefix_tokenizer
68	common_suffix_tokenizer
69	html_tag_remover_tokenizer
70	special_character_tokenizer

ID	Tokenizer Name
71	nested_noun_tokenizer
72	multi_language_tokenizer
73	passive_sentence_tokenizer
74	multi_word_expression_tokenizer
75	sentence_length_tokenizer
76	ignore_numbers_tokenizer
77	contextual_length_tokenizer
78	frequency_contextual_tokenizer
81	content_length_filter_tokenizer
82	emoji_filter_tokenizer
83	abbreviation_tokenizer
84	sentence_based_tokenizer
86	specific_vowel_tokenizer
87	remove_special_character_tokenizer
89	regex_based_tokenizer
90	special_character_removal_tokenizer
91	text_without_numbers_tokenizer
92	specific_length_sentence_tokenizer
93	verb_tokenizer
94	adjective_tokenizer
95	words_with_special_character_tokenizer
99	word_length_tokenizer
100	nested_sentence_tokenizer
104	punctuation_based_tokenizer
105	ignore_case_tokenizer
106	single_word_tokenizer
107	multi_phrase_tokenizer
110	filter_by_length_tokenizer
111	phrase_based_tokenizer
112	content_based_tokenizer
113	adjective_based_tokenizer
114	specific_punctuation_sentence_tokenizer
115	special_character_filter_tokenizer
116	noun_based_tokenizer
117	compound_sentence_tokenizer
118	mixed_case_words_tokenizer
119	ignore_digits_special_tokenizer
121	ignore_common_words_tokenizer
122	ignore_digits_only_tokenizer
123	single_character_filter_tokenizer
125	contextual_noun_tokenizer
126	special_character_based_tokenizer
128	ignore_specific_word_tokenizer
129	punctuation_based_sentence_tokenizer
130	ignore_capitalized_words_tokenizer

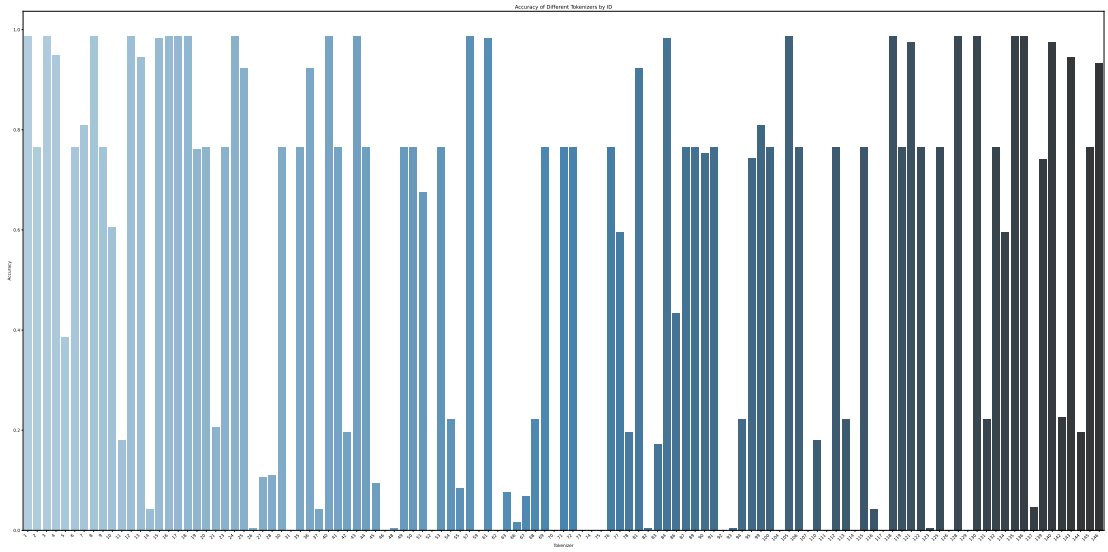
ID	Tokenizer Name
131	contextual_adverb_tokenizer
132	nested_expression_tokenizer
134	ignore_long_words_tokenizer
135	ignore_common_expression_tokenizer
136	sentence_character_tokenizer
137	noun_and_verb_tokenizer
139	filter_long_words_tokenizer
140	ignore_frequent_words_tokenizer
142	vowel_end_tokenizer
143	ignore_noun_words_tokenizer
144	frequency_expression_tokenizer
145	regular_word_tokenizer
146	parsivar_tokenizer

Table 2: Performance Metrics

ID	Precision	Recall	F1	Accuracy
1	0.99	1	0.99	0.99
2	0.87	0.86	0.87	0.77
3	0.99	1	0.99	0.99
4	0.99	0.96	0.97	0.95
5	0.58	0.54	0.56	0.39
6	0.87	0.86	0.87	0.77
7	0.99	0.82	0.89	0.81
8	0.99	1	0.99	0.99
9	0.87	0.86	0.87	0.77
10	0.99	0.61	0.75	0.61
11	1	0.18	0.3	0.18
12	0.99	1	0.99	0.99
13	0.99	0.95	0.97	0.94
14	1	0.04	0.08	0.04
15	0.99	0.99	0.99	0.98
16	0.99	1	0.99	0.99
17	0.99	1	0.99	0.99
18	0.99	1	0.99	0.99
19	0.87	0.86	0.87	0.76
20	0.87	0.86	0.87	0.77
21	1	0.21	0.34	0.21
23	0.87	0.86	0.87	0.77
24	0.99	1	0.99	0.99
25	0.99	0.93	0.96	0.92
26	0.01	0.01	0.01	0
27	1	0.11	0.19	0.11
28	1	0.11	0.2	0.11

ID	Precision	Recall	F1	Accuracy
30	0.87	0.86	0.87	0.77
31	0	0	0	0
35	0.87	0.86	0.87	0.77
36	0.99	0.93	0.96	0.92
37	1	0.04	0.08	0.04
40	0.99	1	0.99	0.99
41	0.87	0.86	0.87	0.77
42	0.98	0.2	0.33	0.2
43	0.99	1	0.99	0.99
44	0.87	0.86	0.87	0.77
45	1	0.09	0.17	0.09
46	0	0	0	0
48	0.03	0	0.01	0
49	0.87	0.86	0.87	0.77
50	0.87	0.86	0.87	0.77
51	0.79	0.82	0.81	0.68
52	0	0	0	0
53	0.87	0.86	0.87	0.77
54	1	0.22	0.36	0.22
55	0.95	0.09	0.16	0.09
57	0.99	1	0.99	0.99
59	0	0	0	0
61	0.99	0.99	0.99	0.98
62	0	0	0	0
63	0.95	0.08	0.14	0.08
66	0.8	0.02	0.03	0.02
67	1	0.07	0.13	0.07
68	1	0.22	0.36	0.22
69	0.87	0.86	0.87	0.77
70	0	0	0	0
71	0.87	0.86	0.87	0.77
72	0.87	0.86	0.87	0.77
73	0	0	0	0
74	0	0	0	0
75	0	0	0	0
76	0.87	0.86	0.87	0.77
77	0.99	0.6	0.75	0.6
78	0.98	0.2	0.33	0.2
81	0.99	0.93	0.96	0.92
82	0.03	0	0.01	0
83	0.79	0.18	0.29	0.17
84	0.99	0.99	0.99	0.98
86	0.88	0.46	0.61	0.43
87	0.87	0.86	0.87	0.77
89	0.87	0.86	0.87	0.77

ID	Precision	Recall	F1	Accuracy
90	0.86	0.86	0.86	0.75
91	0.87	0.86	0.87	0.77
92	0	0	0	0
93	1	0	0.01	0
94	1	0.22	0.36	0.22
95	0.84	0.86	0.85	0.74
99	0.99	0.82	0.89	0.81
100	0.87	0.86	0.87	0.77
104	0	0	0	0
105	0.99	1	0.99	0.99
106	0.87	0.86	0.87	0.77
107	0	0	0	0
110	1	0.18	0.3	0.18
111	0	0	0	0
112	0.87	0.86	0.87	0.77
113	1	0.22	0.36	0.22
114	0	0	0	0
115	0.87	0.86	0.87	0.77
116	1	0.04	0.08	0.04
117	0	0	0	0
118	0.99	1	0.99	0.99
119	0.87	0.86	0.87	0.77
121	0.99	0.98	0.99	0.97
122	0.87	0.86	0.87	0.77
123	1	0	0.01	0
125	0.87	0.86	0.87	0.77
126	0	0	0	0
128	0.99	1	0.99	0.99
129	0	0	0	0
130	0.99	1	0.99	0.99
131	1	0.22	0.36	0.22
132	0.87	0.86	0.87	0.77
134	0.99	0.6	0.75	0.6
135	0.99	1	0.99	0.99
136	0.99	1	0.99	0.99
137	1	0.05	0.09	0.05
139	0.99	0.74	0.85	0.74
140	0.99	0.98	0.99	0.97
142	0.86	0.24	0.37	0.23
143	0.99	0.95	0.97	0.94
144	0.98	0.2	0.33	0.2
145	0.87	0.86	0.87	0.77
146	0.97	0.97	0.97	0.93



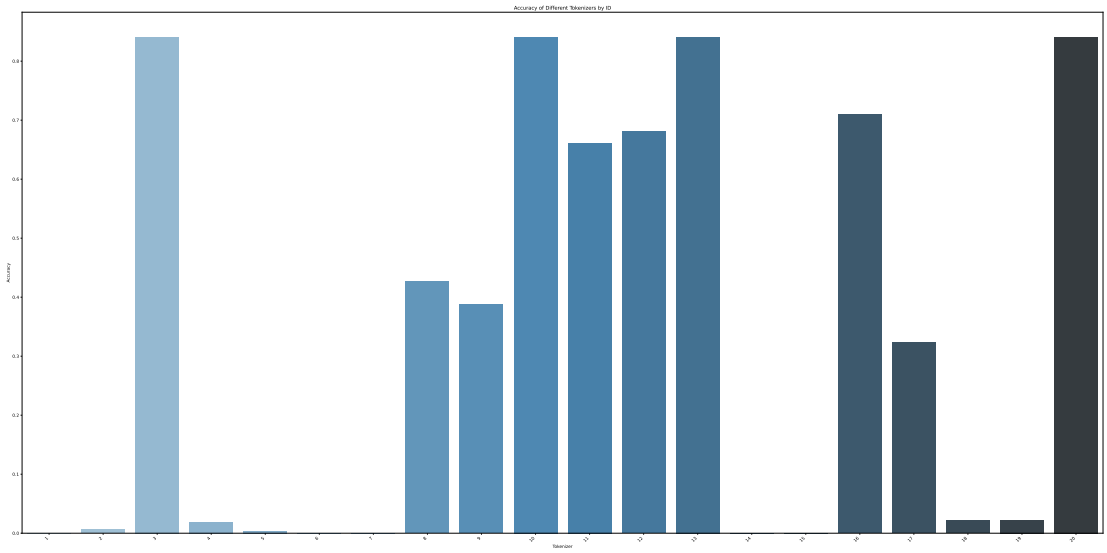
4.1 Challenge 2

In this challenge, all Persian text was connected without any spaces.

Table 3: Tokenizer ID and Name Mapping

ID	Tokenizer Name
1	basic_word_tokenizer
2	regex_persian_word_tokenizer
3	whitespace_tokenizer
4	ngram_tokenizer
5	character_tokenizer
6	stemmer_tokenizer
7	reverse_word_tokenizer
8	prefix_match_tokenizer
9	suffix_match_tokenizer
10	maximum_matching_tokenizer
11	minimum_matching_tokenizer
12	frequency_based_tokenizer
13	prefix_suffix_match_tokenizer
14	regex_pattern_tokenizer
15	hazm_tokenizer
16	probabilistic_tokenizer
17	mixed_tokenizer
18	overlapping_ngram_tokenizer
19	window_sliding_tokenizer
20	recursive_tokenizer

ID	Precision	Recall	F1	Accuracy
1	0	0	0	0
2	0.02	0.01	0.01	0.01
3	0.88	0.95	0.91	0.84
4	0.02	0.12	0.04	0.02
5	0.02	0	0.01	0
6	0	0	0	0
7	0	0	0	0
8	0.49	0.76	0.6	0.43
9	0.46	0.71	0.56	0.39
10	0.88	0.95	0.91	0.84
11	0.85	0.75	0.8	0.66
12	0.85	0.77	0.81	0.68
13	0.88	0.95	0.91	0.84
14	0	0	0	0
15	0	0	0	0
16	0.86	0.8	0.83	0.71
17	0.33	0.95	0.49	0.32
18	0.03	0.06	0.04	0.02
19	0.03	0.07	0.04	0.02
20	0.88	0.95	0.91	0.84



4.2 Challenge 3

In Challenge 3, spaces are randomly added between characters and words based on the statistical modeling and the following algorithm:

Algorithm 2 Add Random Spaces

Input: $T, P_{\text{char}}, P_{\text{word}}, a_{\text{char}}, b_{\text{char}}, a_{\text{word}}, b_{\text{word}}$
Output: processed_text
Normalize T to T_{norm}
Split T_{norm} into tokens
for each token in tokens **do**
    if token is whitespace **then**
        $r \leftarrow$ random()
        if r ¡ P_{word} **then**
            Add S_{word} spaces
        else
            Add 1 space
        end if
    else
        Initialize new_word $\leftarrow \emptyset$
        for char in token **do**
            Append char to new_word
            $r \leftarrow$ random()
            if r ¡ P_{char} **then**
                Add S_{char} spaces
            end if
        end for
        Append new_word to processed_text
        $r \leftarrow$ random()
        if r ¡ P_{word} **then**
            Add S_{word} spaces
        else
            Add 1 space
        end if
    end if
end for
return processed_text

Statistical Formula

Let:

$$
\begin{aligned}
P_{\text{char}} &= \text{Probability of adding spaces between characters,} \\
P_{\text{word}} &= \text{Probability of adding spaces between words,} \\
S_{\text{char}} &\sim U(a_{\text{char}}, b_{\text{char}}), \quad \text{Number of spaces between characters,} \\
S_{\text{word}} &\sim U(a_{\text{word}}, b_{\text{word}}), \quad \text{Number of spaces between words.}
\end{aligned}
$$

The expected number of spaces added between characters is:

$$E[S_{\text{char}}] = \frac{a_{\text{char}} + b_{\text{char}}}{2}$$

The expected number of spaces added between words is:

$$E[S_{\text{word}}] = \frac{a_{\text{word}} + b_{\text{word}}}{2}$$

The algorithm inserts spaces between characters and words based on these probabilities and the uniform distribution for spaces.

4.2.1 Statistical Model

Let:

$$
\begin{aligned}
T &: \text{Input text} \\
P_{\text{char}} &\sim \mathcal{N}(\mu_{\text{char}}, \sigma^2_{\text{char}}) \\
P_{\text{word}} &\sim \mathcal{N}(\mu_{\text{word}}, \sigma^2_{\text{word}}) \\
S_{\text{char}} &\sim U(a_{\text{char}}, b_{\text{char}}) \\
S_{\text{word}} &\sim U(a_{\text{word}}, b_{\text{word}})
\end{aligned}
$$

$$
\begin{aligned}
E[S_{\text{total}}] &= \sum_{i=1}^{n} E[S^i_{\text{char}}] + \sum_{j=1}^{m} E[S^j_{\text{word}}] \\
E[S_{\text{char}}] &= \frac{a_{\text{char}} + b_{\text{char}}}{2} \\
E[S_{\text{word}}] &= \frac{a_{\text{word}} + b_{\text{word}}}{2}
\end{aligned}
$$

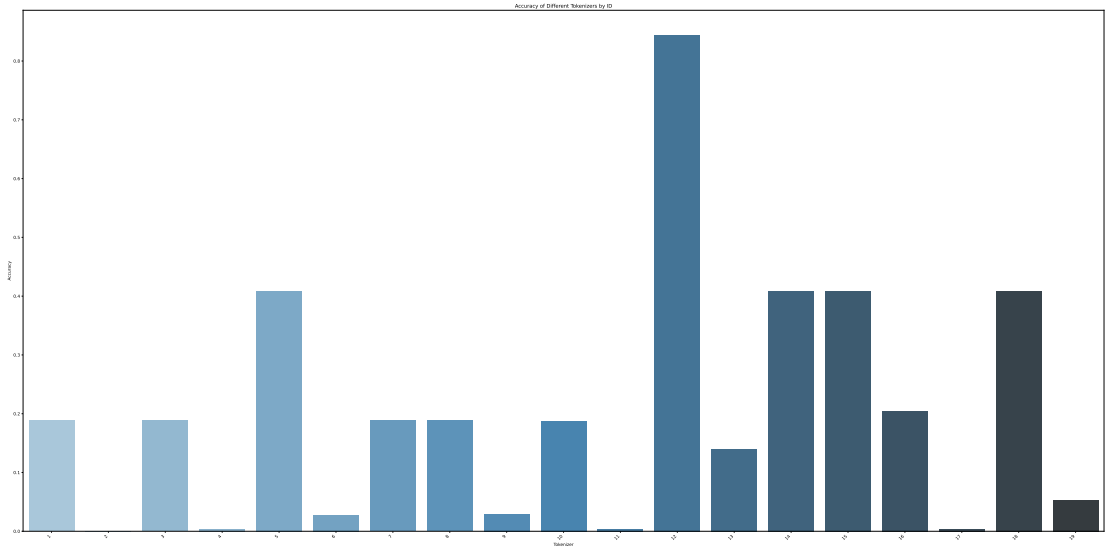
Table 5: Evaluation Metrics

ID	Precision	Recall	F1	Accuracy
1	0.25	0.44	0.32	0.19

ID	Precision	Recall	F1	Accuracy
2	0	0	0	0
3	0.25	0.44	0.32	0.19
4	0.02	0	0.01	0
5	0.74	0.48	0.58	0.41
6	0.05	0.06	0.05	0.03
7	0.25	0.44	0.32	0.19
8	0.25	0.44	0.32	0.19
9	0.05	0.06	0.06	0.03
10	0.25	0.44	0.32	0.19
11	0.02	0	0.01	0
12	0.88	0.95	0.92	0.84
13	0.17	0.44	0.25	0.14
14	0.74	0.48	0.58	0.41
15	0.74	0.48	0.58	0.41
16	0.27	0.47	0.34	0.2
17	0.02	0	0.01	0
18	0.74	0.48	0.58	0.41
19	0.08	0.14	0.1	0.05

Table 6: Tokenizer ID and Name Mapping

ID	Tokenizer Name
1	multiple_spaces_split
2	remove_all_spaces
3	normalize_spaces
4	random_spaces_ignoring
5	space_inclusive_dictionary_match
6	space_based_ngram
7	space_aware_regex
8	conditional_space_split
9	fixed_window_space_handling
10	alternating
11	skip_spaces
12	space_normalization_dictionary
13	hybrid_space_ngram
14	space_based_prefix_match
15	space_suffix_match
16	hazm_space_normalized
17	space_character_filtering
18	space_based_maximum_matching
19	space_heuristic_rule



4.3 Challenge 4

Really challenging Persian text which includes word mistakes, informal text, bad written words, and.... were generated with LLM and tested

ID	Precision	Recall	F1	Accuracy
1	0.71	0.79	0.75	0.6
2	0.82	0.88	0.85	0.74
3	0.71	0.79	0.75	0.6
4	0.71	0.76	0.73	0.58
5	0.44	0.46	0.45	0.29
6	0.82	0.88	0.85	0.74
7	0.63	0.51	0.56	0.39
8	0.71	0.79	0.75	0.6
9	0.82	0.88	0.85	0.74
10	0.53	0.27	0.35	0.21
11	0.92	0.28	0.43	0.28
12	0.71	0.79	0.75	0.6
13	0.71	0.76	0.73	0.58
14	0.9	0.06	0.1	0.06
15	0.83	0.9	0.87	0.76
16	0.71	0.79	0.75	0.6
17	0.71	0.79	0.75	0.6
18	0.71	0.79	0.75	0.6
19	0.81	0.88	0.85	0.73
20	0.83	0.9	0.86	0.76
21	0.8	0.24	0.37	0.23

ID	Precision	Recall	F1	Accuracy
23	0.82	0.88	0.85	0.74
24	0.71	0.79	0.75	0.6
25	0.7	0.7	0.7	0.54
26	0.02	0.02	0.02	0.01
27	0.67	0.05	0.09	0.05
28	1	0.09	0.16	0.09
30	0.82	0.88	0.85	0.74
31	0	0	0	0
35	0.82	0.88	0.85	0.74
36	0.7	0.7	0.7	0.54
37	0.9	0.06	0.1	0.06
40	0.71	0.79	0.75	0.6
41	0.82	0.88	0.85	0.74
42	0.79	0.23	0.36	0.22
43	0.71	0.79	0.75	0.6
44	0.91	0.95	0.93	0.87
45	1	0.06	0.12	0.06
46	0	0	0	0
48	0	0	0	0
49	0.82	0.88	0.85	0.74
50	0.82	0.88	0.85	0.74
51	0.82	0.9	0.86	0.76
52	0	0	0	0
53	0.82	0.88	0.85	0.74
54	0.88	0.09	0.16	0.09
55	0.82	0.06	0.1	0.05
57	0.71	0.79	0.75	0.6
59	0.04	0.01	0.02	0.01
61	0.72	0.79	0.75	0.6
62	0	0	0	0
63	0.85	0.1	0.19	0.1
66	0.8	0.02	0.05	0.02
67	0.33	0.01	0.01	0.01
68	0.88	0.09	0.16	0.09
69	0.82	0.88	0.85	0.74
70	0	0	0	0
71	0.82	0.88	0.85	0.74
72	0.91	0.95	0.93	0.87
73	0	0	0	0
74	0.04	0.01	0.02	0.01
75	0	0	0	0
76	0.82	0.88	0.85	0.74
77	0.76	0.68	0.72	0.56
78	0.79	0.23	0.36	0.22
81	0.7	0.7	0.7	0.54

ID	Precision	Recall	F1	Accuracy
82	0	0	0	0
83	0.87	0.33	0.48	0.31
84	0.83	0.9	0.87	0.76
86	0.94	0.36	0.52	0.36
87	0.82	0.88	0.85	0.74
89	0.91	0.95	0.93	0.87
90	0.93	0.95	0.94	0.89
91	0.82	0.88	0.85	0.74
92	0	0	0	0
94	0.88	0.09	0.16	0.09
95	0.71	0.78	0.74	0.59
98	0	0	0	0
99	0.63	0.51	0.56	0.39
100	0.91	0.95	0.93	0.87
104	0	0	0	0
105	0.71	0.79	0.75	0.6
106	0.91	0.95	0.93	0.87
107	0.04	0.01	0.02	0.01
110	0.92	0.28	0.43	0.28
111	0.04	0.01	0.02	0.01
112	0.82	0.88	0.85	0.74
113	0.88	0.09	0.16	0.09
114	0	0	0	0
115	0.82	0.88	0.85	0.74
116	1	0.01	0.01	0.01
117	0	0	0	0
118	0.71	0.79	0.75	0.6
119	0.82	0.88	0.85	0.74
121	0.71	0.78	0.75	0.6
122	0.82	0.88	0.85	0.74
123	0	0	0	0
125	0.82	0.88	0.85	0.74
126	0	0	0	0
128	0.71	0.79	0.75	0.6
129	0	0	0	0
130	0.71	0.79	0.75	0.6
131	0.88	0.09	0.16	0.09
132	0.82	0.88	0.85	0.74
134	0.76	0.68	0.72	0.56
135	0.71	0.79	0.75	0.6
136	0.71	0.79	0.75	0.6
137	1	0.01	0.01	0.01
139	0.75	0.75	0.75	0.6
140	0.71	0.78	0.75	0.6
142	0.84	0.19	0.31	0.18

ID	Precision	Recall	F1	Accuracy
143	0.71	0.78	0.74	0.59
144	0.79	0.23	0.36	0.22
145	0.82	0.88	0.85	0.74
146	0.88	0.93	0.9	0.82

Tokenizer ID and Name Mapping LaTeX table:

ID	Tokenizer Name
1	basic_word
2	regex_persian_word
3	whitespace
4	stopword_removal
5	stemmer
6	punctuation_removal
7	length_based
8	unique_word
9	number_aware
10	long_words
11	short_word
12	reverse_word
13	specific_word_removal
14	frequency
15	sentence_based_word
16	bigram
17	trigram
18	ngram
19	persian_and_punctuation
20	sentence_then_word
21	specific_length
23	ignore_digits
24	frequency_count
25	ignore_short_words
26	reverse_each_word
27	startswith
28	endswith
30	ignore_punctuation
31	long_sentence
35	persian_and_digits
36	remove_short_words
37	frequent_words
40	first_char_length
41	alphabetical
42	repeated_words
43	case_insensitive

ID	Tokenizer Name
44	ignore_special_char
45	endswith_specific_char
46	short_sentence
48	character
49	count_vectorizer
50	normalized
51	ParsBert
52	exclamatory_sentence
53	contextual_word
54	adverb
55	interjection
57	word_count
59	phrase
61	content_word
62	repeated_character
63	frequency_filter
66	random_sampling
67	common_prefix
68	common_suffix
69	html_tag_remover
70	special_character
71	nested_noun
72	multi_language
73	passive_sentence
74	multi_word_expression
75	sentence_length
76	ignore_numbers
77	contextual_length
78	frequency_contextual
81	content_length_filter
82	emoji_filter
83	abbreviation
84	sentence_based
86	specific_vowel
87	remove_special_character
89	regex_based
90	special_character_removal
91	text_without_numbers
92	specific_length_sentence
94	adjective
95	words_with_special_character
98	passive_voice_sentence
99	word_length
100	nested_sentence
104	punctuation_based

ID	Tokenizer Name
105	ignore_case
106	single_word
107	multi_phrase
110	filter_by_length
111	phrase_based
112	content_based
113	adjective_based
114	specific_punctuation_sentence
115	special_character_filter
116	noun_based
117	compound_sentence
118	mixed_case_words
119	ignore_digits_special
121	ignore_common_words
122	ignore_digits_only
123	single_character_filter
125	contextual_noun
126	special_character_based
128	ignore_specific_word
129	punctuation_based_sentence
130	ignore_capitalized_words
131	contextual_adverb
132	nested_expression
134	ignore_long_words
135	ignore_common_expression
136	sentence_character
137	noun_and_verb
139	filter_long_words
140	ignore_frequent_words
142	vowel_end
143	ignore_noun_words
144	frequency_expression
145	regular_word
146	parsivar

5 References

[1]Kamali, D., Janfada, B., Shenasa, M. E., Minaei-Bidgoli, B. (2022). Evaluating Persian Tokenizers. arXiv preprint arXiv:2202.10879.

[2] Etezadi, R., Karrabi, M., Zare, N., Sajadi, M. B., Pilehvar, M. T. (2022, July). Dadmatools: Natural language processing toolkit for persian language. In Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: System Demonstrations (pp. 124-130).

[3]Dagan, G., Synnaeve, G., Rozière, B. (2024). Getting the most out of your tokenizer for pre-training and domain adaptation. arXiv preprint arXiv:2402.01035.

[4]Petrov, A., La Malfa, E., Torr, P., Bibi, A. (2024). Language model tokenizers introduce unfairness between languages. Advances in Neural Information Processing Systems, 36.

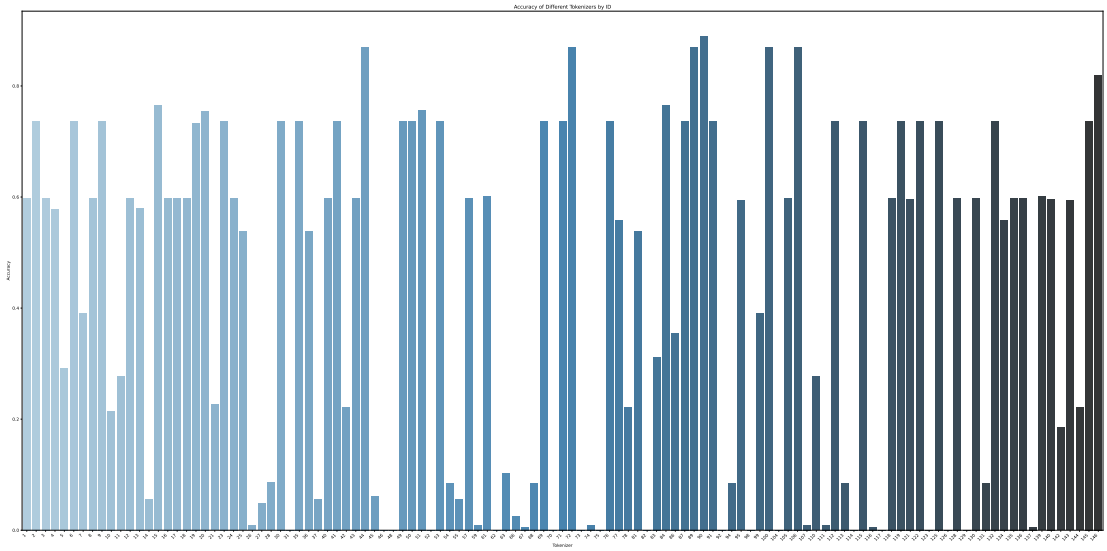


Figure 1: Challenge 4